YOLO-Based Face Mask Detection on Low-End Devices Using Pruning and Quantization

Benedetta Liberatori*, Ciro Antonio Mami*, Giovanni Santacatterina*,
Marco Zullich**, and Felice Andrea Pellegrino**

*Department of Mathematics and Geosciences, University of Trieste, Italy
{benedetta.liberatori, ciroantonio.mami, giovanni.santacatterina}@studenti.units.it
**Department of Engineering and Architecture, University of Trieste, Italy
marco.zullich@phd.units.it, *fapellegrino@units.it*

***Abstract*—Deploying Deep Learning (DL) based object detection (OD) models in low-end devices, such as single board computers, may lead to poor performance in terms of frames-per-second (FPS). Pruning and quantization are well-known compression techniques that can potentially lead to a reduction of the computational burden of a DL model, with a possible decrease of performance in terms of detection accuracy. Motivated by the widespread introduction of face mask mandates by many institutions during the Covid-19 pandemic, we aim at training and compressing an OD model based on YOLOv4 to recognize the presence of face masks, to be deployed on a Raspberry Pi 4. We investigate the capability of different kinds of pruning and quantization techniques of increasing the FPS with respect to the uncompressed model, while retaining the detection accuracy. We quantitatively assess the pruned and quantized models in terms of Mean Average Precision (mAP) and FPS, and show that with proper pruning and quantization, the FPS can be doubled with a moderate loss in mAP. The results provide guidelines for compression of other OD models based on YOLO.**

***Keywords*—Deep Learning; Object Detection; YOLO; TinyML; Face mask detection; Pruning; Quantization.**

I. Introduction

As modern deep neural networks grow more complex, their computation and memory requirements emerge as an obstacle, preventing their deployment on resource-constrained embedded and mobile devices. Vision has become an important part of many smart embedded systems. Object Detection (OD), in particular, is a visual task linked to many real-word applications, such as video surveillance and autonomous driving. Current state-of-the-art OD models have millions of parameters, which renders their implementation on low-end or embedded devices quite a challenge. Since the World Health Organization stated that medical masks are effective in preventing the spread of Covid-19, it has become important, and in some cases even mandatory, to wear them in public areas. Thus, a reliable and fast AI solution for detecting whether people in a specific area are or are not wearing a face mask could be of great use, given the difficulty to manually monitor each individual. Moreover, considering the hardware and power limitations of the devices which are usually employed for video surveillance tasks, the need to ensure that the OD pipeline runs with enough speed on such equipment arises. Our aim is to train an object detector to reliably perform face mask detection on a low-end device. It follows that the accuracy of the model is not our sole performance indicator; rather, we look for a good compromise between detection accuracy and speed of execution. We wish our model to be run reliably even on some devices with low computational capability and no Graphics Processing Unit (GPU). We start from a lightweight implementation of YOLOv4 [1] introduced in [2]; we apply *filter pruning* [3] in order to achieve the aforementioned objective. Filter pruning operates a sparsification of the Artificial Neural Network (ANN) parameters via the removal of entire convolutional filters according to a given criterion, speeding up the computation. After pruning, we also perform *quantization* (i.e., reduction of the number of bits used to represent each parameter), obtaining an additional improvement in the speed of inference, at the cost of a small reduction in detection accuracy. We deploy the model on a Raspberry Pi 4, achieving fast inference and satisfactory accuracy results. The adoption of this device is motivated by the necessity of using a low-cost device, such as a single-board computer which has the capability of running ANN-based Computer Vision tasks. The Raspberry Pi 4 has a widespread usage, as indicated by the extensive literature reporting applications of such a device in various technical and scientific fields (see for instance [4] and the references therein). This enables the detector to potentially be used in a wide number of real-world applications where high-end systems or constant human surveillance would be unavailable or incongruous, such as in shops or offices. In order to assess the performance of our model, we take into consideration two indicators: Mean Average Precision (mAP), measuring the detection accuracy, and frames-per-second (FPS), measuring the number of images (*frames*) that the model is able to process at inference time. We record FPS on the Raspberry Pi 4. The goal is the following: we wish to optimize the FPS, without losing too much mAP. While the original YOLOv4-based model achieves 0.99 FPS with a mAP of 0.618, we are able to effectively prune and quantize the model, reaching 1.97 FPS—improving this metric by 99%—and 0.574 mAP—a 7% decrease with respect to the original model. We point out that, in the present

work, we do not introduce technical novelties, neither we operate architectural modifications of the ANN we make use of. Instead, given a pre-existing architecture, we showcase that, with the application of pruning and quantization, it is possible to record a large increase in FPS without hurting mAP too much, thus aiding the deployment of the model on low-end devices. We release the code of our implementation to the following GitHub repository: https://github.com/benedettaliberatori/Modified-Yolov4Tiny-RaspberryPi/.

II. Related work

A. Object Detection (OD)

OD consists of identifying and locating instances of objects from a particular class within a video or image. The locations of said objects are roughly determined, and a bounding shape is drawn around each object. There exists a handful of ANN-based architectures for performing OD: RCNN [5] and its variants, YOLO [6] and its variants, RetinaNet [7], and more recent transformer-based architectures like SWIN Transformers [8]. Despite the latter having recorded state-of-the-art results on common OD benchmarks, YOLO-based solutions find widespread applicability due to having lower computational requirements, while still recording acceptable levels of detection accuracy [9, 10]. These models, despite being very fast if compared to their alternatives, are still not suited to be ran on inexpensive single-board computers, such as Raspberry Pi's. For this reason, lightweight versions of YOLO have been developed, like [2, 11], by operating tweaks in their architecture, e.g., decreasing the number of *detecting heads*, i.e., number of different scales at which predictions are made, or lowering the depth of convolutional layers. As a result, the running speed is significantly increased but detection accuracy is reduced.

1) Face mask detection: Recently, due to the Covid-19 pandemic, many works have been published devoted to the automatic recognition of face masks. There does not seem to exist, though, a *de-facto* dataset for benchmarking the task of OD applied to face masks. A handful of existing datasets, along with their pros and cons, have been reported in [12]; some works cited in the present paper employ their own dataset, which is sometimes not even publicly accessible (as in [13]). We are not aware of other works using the same dataset as ours. A large body of work in this area is targeting high levels of accuracy, disregarding the computational requirements of the proposed solutions [12, 13, 14, 15], or assessing the speed of inference on high-end hardware [16, 17], overlooking deployment on low-end devices. Kong et al. [18] proposed the solution which, to the best of our knowledge, comes closer to ours when the objectives of the work are concerned: they developed a two-stage lightweight model for first operating face detection, then performing classification within the predicted bounding boxes to determine the presence or absence of the face mask on the identified face. They also deploy their model on a Raspberry Pi 4, as we do, although they enhance the device with a Neural Compute Stick to speed-up the tensor operations, which we do not do.

B. ANN compression

Some Machine Learning models have large memory requirements [19]. Pruning and quantization—which we make use of in the present work—are possible solutions for reducing these requirements, but knowledge distillation [20] and tensor/matrix decomposition [21] are also viable alternatives.

ANN pruning acts by setting to zero individual parameters of the model. Specifically, it can be divided into *structured* and *unstructured* pruning [22]. The latter prunes parameters without concern for the geometry of the layers, while the former removes whole groups of weights, such full convolutional filters in the case of Convolutional Neural Networks (CNNs). In this last case, we can also talk of *filter pruning*. Structured pruning can lead to increased inference speed without needing required libraries or hardware [23], thus it can be used to produce lightweight ANNs for deployment on non-specialized low-end devices, like the aforementioned Raspberry Pi's. A widely used technique for filter pruning introduced by Li et al. [3] consists of pruning filters which, after a first *pre-training* phase, exhibit a small L1 norm compared to the other filters within the layer(s). The rationale behind it is that filters with a lot of small-magnitude parameters may be unimportant to the production of accurate predictions: this is an example of *magnitude-based post-train pruning*. In this context, immediately after the pruning step, the CNN often records sub-par accuracy, which calls for a *re-training* phase. This gives way to an *iterative* application of pruning [24], in which a training phase is followed by pruning, which is followed by re-training, and so on. The re-training is called *fine-tuning* when it is performed for a smaller number of epochs than the original training [25]. In the literature, it has not yet been established whether fine-tuning or full re-training leads to the best results in terms of accuracy, with somehow inconsistent conclusions [23, 25, 26, 27]. For that reason, in this work, we will be using fine-tuning, as it requires fewer iterations with respect to full re-training.

ANN quantization [28] consists in using a lower-bit representation of the parameters to store real-valued weights and activations. Most ANNs are trained using floating-point double precision (FP32), which in some cases can be more than needed: accurate results can be obtained also by employing half-precision floating-point (FP16) [29] or integer arithmetic (INT16 to INT4) [30]. This approach can be used both at training time (*quantization-aware training*), and at inference time (*post-training quantization*) [31].

III. Materials and Methods

For the task of face mask detection, we seek a good trade-off between computational speed and accuracy, thus resorting to YOLO, specifically its version 4 [1]. YOLO divides the input image into an $S \times S$ grid. Each cell within the grid outputs a prediction at different scales. The

dimensions of the predicted bounding boxes are biased towards some pre-computed values, which are referred to as *anchor boxes*. In the present work, we use the YOLOv4-tiny variant introduced by Jiang et al. [2]. They replaced the two CSPBlock [32] modules of YOLO-v4 with two ResBlock-D modules [33], which require much fewer floating-point operations to compute, and introduced the Convolutional Block Attention Module, which is used to realize spatial-wise and channel-wise attention. The loss function is composed of four parts:

$$\mathcal{L}\text{oss} = \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{no-obj}}\mathcal{L}_{\text{no-obj}} + \lambda_{\text{obj}}\mathcal{L}_{\text{obj}} + \lambda_{\text{class}}\mathcal{L}_{\text{class}}. \quad (1)$$

The $\mathcal{L}_{\text{box}}$ loss is a mean squared error loss which measures, for each prediction, how different the predicted box and its corresponding ground truth are. The $\mathcal{L}_{\text{no-obj}}$, $\mathcal{L}_{\text{obj}}$, and $\mathcal{L}_{\text{class}}$ loss are all based on the Binary Cross-Entropy (BCE) function and are used to guide the model to correctly predict the presence or absence of an object and to drive said prediction to the correct class.

We train the model using the Rectified-Adam (RAdam) [34] optimizer, designed to tackle the fact that, with regular Adam [35], the adaptive learning rate in general suffers from high variance in the early stages, thus requiring expensive warm-up phases.

A. ANN pruning

We employ the method proposed in [3], which consists in pruning less relevant filters in CNNs. It is a structured technique which determines the parameters to be pruned by ranking the filters within each layer according to their L1-norm. Then, a fixed number of low-norm filters is deleted from each layer. The number of filters to be pruned is determined by a hyperparameter which is called *pruning rate*.

B. Conversion to TFLite and Quantization

We train our model on Python using PyTorch [36] version 1.8. Subsequently, we convert the model to Tensorflow using ONNX[1], and finally to Tensorflow Lite (TFLite) [37], a DL framework which allows for translating the model from Python to C++ for the purpose of deployment into ARM-based CPU-only devices such as our Raspberry Pi 4. TFLite comes along with several tools for quantization. We experiment with (a) *static* quantization to integer with 8-bit precision with floating-point fallback, and (b) dynamic-range quantization [31]. Both parameters and data are quantized to integer precision. In (a), the data are converted to INT8 according to a fixed pre-computed range, while in (b), the range is re-calculated for each inference cycle. This can result in a lower inference speed (due to the recalculation of ranges), but may provide more accurate models, as the data conversion to INT8 is less prone to distortions. The keyword *floating-point fallback*[2], refers to the possibility that some computations may still be ran on floating-point precision when they do not have an equivalent integer-arithmetic implementation within the library.

IV. Experimental settings and results

A. Dataset

For our experiments, we used a publicly available dataset called "Mask-Detection-Dataset"[3], composed of images containing various examples of people with and without face masks. The dataset is built in such a way that the people appear at different scales and in different scenarios. The training set and test set contain, respectively, 5448 and 1318 frames. We computed the candidate *anchor boxes* starting from the bounding boxes dimensions found within the ground truth of the dataset. We did so via k-means clustering, with $k = 6$ number of centroids, using intersection-over-union (IoU) as distance metric. Thus, six different boxes are obtained: three for each scale. We then resized the input frames to the resolution of 416×416 pixels and performed *data augmentation*, applying random rotation (with probability 0.5) between $[-20°, 20°]$ and horizontal flipping (still with probability 0.5). Moreover, we operated a *preprocessing step* aiming at degrading the quality of the images by down-scaling by a factor of 75%, then applying motion blur[4] with kernel size uniformly sampled between 3 and 7.

B. Training

The model was pre-trained for 100 epochs, with learning rate (LR) of 2×10^{-4}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. Moreover, we employed weight decay with a coefficient of 0.005 and used a batch size of 32. Loss parameters λ_{box}, λ_{noobj}, λ_{obj} and λ_{class} from Equation (1) were respectively set to 1, 5, 10 and 1. In order to optimize the LR, we ran a grid search over the values 0.001, 5×10^{-4}, 2×10^{-4}, and 1×10^{-4}, obtaining best results for the value 2×10^{-4}, which, coincidentally, is very close to the ratio, suggested in [38], between the default LR of 0.001 and the square root of the batch size. As previously stated, the model was implemented in Pytorch version 1.8 and trained using an NVidia V100 GPU.

C. Pruning and Quantization

We experimented with two different structured pruning procedures: (a) one-shot pruning with fine-tuning, and (b) iterative pruning with learning rate rewind [25] coupled with fine-tuning. As stated in Section III, we selected filters to be pruned with respect to their L1-norm, retraining for 50 epochs in the one-shot setting and for 5 in the iterative one, with 7 iterations. The *pruning rate* is tuned between 0.5, 0.6, 0.7, 0.8 and 0.9 for the one-shot case, while in the iterative case we experiment with 0.1 and 0.2. After pruning, we selected the best model according to mAP and FPS and proceeded with applying post-training quantization. We experimented with both static and dynamic quantization.

[1] https://onnx.ai

[2] https://www.tensorflow.org/lite/performance/post_training_quantization

[3] https://github.com/archie9211/Mask-Detection-Dataset

[4] Using the `augmentation.transforms.MotionBlur` method from the Pyhton library "albumentations" (www.albumentations.io).

TABLE I: NUMBER OF PARAMETERS, FRAMES-PER-SECOND (FPS), AND MEAN AVERAGE PRECISION (MAP) FOR EACH OF THE MODELS AFTER PRUNING IS APPLIED. THE UNPRUNED YOLOV4-TINY MODEL IS INDICATED AS "BASELINE". NAMING CONVENTION AS IN SECTION IV-D2.

model	# parameters	FPS	mAP
baseline	9.1M	0.99	0.618
one-shot$_{0.5}$	3.1M	1.59	0.584
one-shot$_{0.6}$	**2.5M**	**1.67**	**0.587**
one-shot$_{0.7}$	2.1M	1.80	0.489
one-shot$_{0.8}$	1.8M	1.93	0.446
one-shot$_{0.9}$	1.5M	**2.07**	0.233
iterative$_{0.1}$	3.0M	1.51	0.601
iterative$_{0.2}$	1.8M	1.93	0.511

TABLE II: NUMBER OF PARAMETERS, MODEL SIZE IN MB, FPS AND MAP FOR "BASELINE" MODEL (UNPRUNED AND UNQUANTIZED), ONE-SHOT$_{0.6}$ BEFORE AND AFTER QUANTIZATION. STATIC QUANTIZATION WITH FLOATING-POINT FALLBACK IS REFERRED TO AS "ST-Q", WHILE DYNAMIC QUANTIZATION IS CALLED "DYN-Q". NAMING CONVENTION AS IN SECTION IV-D2.

model	# params.	model size (MB)	FPS	mAP
baseline	9.1M	36.5	0.99	0.618
one-shot$_{0.6}$	2.5M	12.3	1.67	0.587
one-shot$_{0.6}$ dyn-q	2.5M	2.7	1.48	0.586
one-shot$_{0.6}$ st-q	**2.5M**	**2.7**	**1.97**	**0.574**

D. Results

1) Evaluation metrics: The various models are evaluated according to two different metrics: (a) Mean Average Precision (mAP) at IoU[5] threshold of 0.5, and (b) frames-per-second (FPS). In other references[6], mAP is also called $AP^{IoU=0.50}$. It measures the detection accuracy taking into account both the correctness of the predicted class and the overlap between the predicted and the ground truth boxes. FPS, instead, measures the inference speed by counting how many single frames are sequentially elaborated by the model during execution on a given machine. We record the FPS on the Raspberry Pi 4.

2) Pruning: In Table I, we report the results in terms of mAP and FPS for the unpruned and the pruned models. The unpruned model is referred to as "baseline", while the pruned models are named according to pruning technique employed, i.e., "one-shot" or "iterative", followed by the pruning rate written in subscript. For instance, one-shot$_{0.5}$ refers to a model sparsified with one-shot pruning (see Section IV-C) with a pruning rate of 0.5. We report the mAP *after* the application of the pre-processing pipeline ("degradation") reported in Section IV-A in an attempt to reproduce the low-fidelity regime in which the model is supposed to be deployed.

Our best result in terms of FPS (2.07) is achieved after a one-shot pruning applied with a 90% pruning rate (one-shot$_{0.9}$), at the expense of a considerable decrease in mAP with respect to the baseline. The model one-shot$_{0.6}$, despite not behaving as well as one-shot$_{0.9}$ in terms of FPS, achieves an acceptable compromise between mAP (0.587) and FPS (1.67). We thus deem one-shot$_{0.6}$ to be the best pruned model and proceed to apply quantization to it.

3) Quantization: Table II presents the results in terms of FPS and mAP for one-shot$_{0.6}$ after post-training quantization. We keep the naming convention introduced in Table I, appending "st-q" for indicating static quantization and "dyn-q" for dynamic quantization. We can witness how dynamic quantization, despite leaving the mAP almost unchanged, loses in FPS with respect to the pruned model. This is most likely due to the fact that the recalculation of dynamic ranges introduces a large computational overhead. On the other hand, static quantization records a small decrease in mAP (from 0.587 to 0.574), with a large increase in FPS (from 1.67 to 1.97). Moreover, quantization sensibly reduces the model size of the pruned model, from 12.3 to 2.7 MB. Thus, we select one-shot$_{0.6}$ with static quantization as the final model for solving the task of face mask detection.

In Figure 1 we showcase the behavior of one-shot$_{0.6}$ with static quantization on a set of on-the-wild images, pointing out both strengths and weaknesses of our model. A complete comment on that is present in Section V-A.

V. CONCLUSION

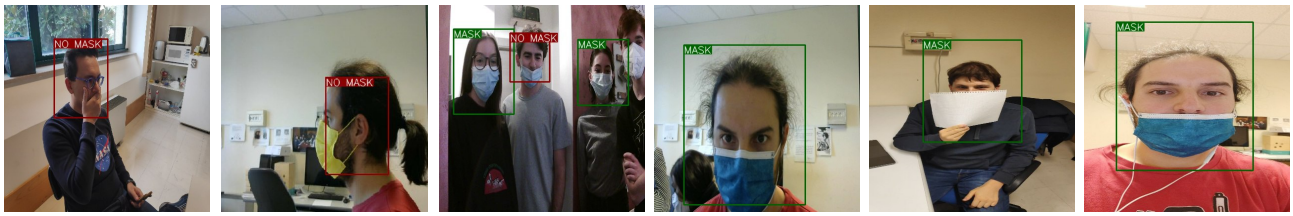
We proposed a lightweight Object Detection (OD) solution for the detection of face masks on people. In order for our pipeline to be exploited in many real-life scenarios, we wanted our model to have very small computational requirements, so as to enable its implementation on low-end devices. Our solution considered as a baseline a variant of YOLOv4-Tiny, originally introduced in [2]. We trained it on a publicly available dataset ("Mask-Detection-Dataset"). We then operated filter pruning, with several re-training strategies, and two techniques for quantization. After having applied (a) one-shot filter pruning with pruning rate 0.6 and (b) static quantization with integer fallback, we obtained a much faster model (1.97 frames-per-second—FPS—compared to 0.99 FPS of the baseline) at the cost of a smaller detection accuracy (0.574 Mean Average Precision—mAP—compared to 0.618 mAP of the baseline); a good trade-off between these two metrics was the target of our work, as stated in Section I.

A. Discussion

A work with similar objectives as ours is [18]: it targets deployment on low-end devices and it tackles the problem of image degradation due to the extremely limited resources of these computers; however, it does not employ YOLO and, instead, makes use of a two-stage pipeline of face detection + face mask classification. A performance comparison with this work is, though, unfeasible as (a) the authors did not release their code publicly, and (b) FPS are reported on a Raspberry Pi 4 enhanced with a neural compute stick, which we do not employ, thus rendering impossible a direct comparison of their reported results

[5]The IoU quantifies the overlap between the predicted and the ground trouth bounding boxes: it is calculated as the ratio between the intersection of the areas of the two bounding boxes and their union.

[6]https://cocodataset.org/#detection-eval



(a) (b) (c) (d) (e) (f)

Fig. 1: Output of our final ANN one some on-the-wild images, showcasing strengths and weaknesses of this model. Preprocessing not applied for clarity.

with ours. Other works reported in the present document do not target an implementation on low-end device—their objectives are substantially different from ours—thus we deem a comparison with their solutions to be out of the scope of our project. We can notice, though, that the mAP reported in some other works seems to be comparable with the one we obtained. For instance, Roy et al. [17] report a mAP of 0.5627, while Kong et al. [18] report a mAP of 0.645, both using YOLOv3. Kumar et al. [12] communicate a mAP of 0.57 using YOLOv4. These values seem to be in line with our reported mAP of 0.574 using the final quantized model.

In Figure 1, we can observe some strengths and weaknesses of our model deployed on-the-wild. First, we can notice that it struggles with small or close objects (see (1d), the female on the left to the correctly identified male), or when faces are close to the edge of the image (1c). These are common YOLO letdowns, already noted in the literature (e.g., see [39]). Moreover, the model tends to struggle when faces are positioned in profile (1b). On the other hand, we can see that occluding one's mouth with a hand is not fooling the model (1a), while occlusion with objects with a color similar to that of a face-mask, e.g., a paper sheet, may produce wrong predictions (1e). Finally, the model achieves mixed results when the mask is worn incorrectly: in (1c) it correctly identifies that the male in the center is wearing the mask under his nose; conversely, in (1f) it does not. This weakness is certainly due to the dataset, which lacks images of incorrectly-worn masks, thus the model does not know how to behave in these situations. Kumar et al. [12], Degadwala et al. [13] have solved this issue by designing datasets whose labels include a third category—mask incorrectly worn.

Ideas for *future work* include (a) extension to other, more comprehensive datasets, and (b) experimenting with more computationally-expensive pruning techniques, like *learning-rate rewind* with full retraining [25], or more recent filter pruning ones, like those proposed in [40] or [41].

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[2] Z. Jiang, L. Zhao, S. Li, and Y. Jia, "Real-time object detection method based on improved yolov4-tiny," *ArXiv*, vol. abs/2011.04244, 2020.

[3] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2017.

[4] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–5.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[8] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," *arXiv preprint arXiv:2103.14030*, 2021.

[9] V. Kharchenko and I. Chyrka, "Detection of airplanes on the ground using yolo neural network," in *2018 IEEE 17th international conference on mathematical methods in electromagnetic theory (MMET)*. IEEE, 2018, pp. 294–297.

[10] L. Tan, T. Huangfu, L. Wu, and W. Chen, "Comparison of retinanet, ssd, and yolo v3 for real-time pill identification," *BMC medical informatics and decision making*, vol. 21, no. 1, pp. 1–11, 2021.

[11] W. Fang, L. Wang, and P. Ren, "Tinier-yolo: A real-time object detection method for constrained environments," *IEEE Access*, vol. 8, pp. 1935–1944, 2020.

[12] A. Kumar, A. Kalia, K. Verma, A. Sharma, and M. Kaushal, "Scaling up face masks detection with yolo on a novel dataset," *Optik*, vol. 239, p. 166744, 2021.

[13] S. Degadwala, D. Vyas, U. Chakraborty, A. R. Dider, and H. Biswas, "Yolo-v4 deep learning model for medical face mask detection," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 2021, pp. 209–213.

[14] J. Yu and W. Zhang, "Face mask wearing detection

algorithm based on improved yolo-v4," *Sensors*, vol. 21, no. 9, p. 3263, 2021.

[15] S. Abbasi, H. Abdi, and A. Ahmadi, "A face-mask detection approach based on yolo applied for a new collected dataset," in *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*, 2021, pp. 1–6.

[16] S. Asif, Y. Wenhui, Y. Tao, S. Jinhai, and K. Amjad, "Real time face mask detection system using transfer learning with machine learning method in the era of covid-19 pandemic," in *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2021, pp. 70–75.

[17] B. Roy, S. Nandy, D. Ghosh, D. Dutta, P. Biswas, and T. Das, "Moxa: a deep learning based unmanned approach for real-time monitoring of people wearing medical masks," *Transactions of the Indian National Academy of Engineering*, vol. 5, no. 3, pp. 509–518, 2020.

[18] X. Kong *et al.*, "Real-time mask identification for covid-19: An edge computing-based deep learning framework," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[19] J. O. Neill, "An overview of neural network compression," *arXiv preprint arXiv:2006.03669*, 2020.

[20] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2n learning: Network to network compression via policy gradient reinforcement learning," *arXiv preprint arXiv:1709.06030*, 2017.

[21] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *Interspeech*, 2013, pp. 2365–2369.

[22] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *arXiv preprint arXiv:2102.00554*, 2021.

[23] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019.

[24] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 1135–1143.

[25] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," *arXiv preprint arXiv:2003.02389*, 2020.

[26] M. Zullich, E. Medvet, F. A. Pellegrino, and A. Ansuini, "Speeding-up pruning for artificial neural networks: introducing accelerated iterative magnitude pruning," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 3868–3875.

[27] S. Liu *et al.*, "Sparse training via boosting pruning plasticity with neuroregeneration," *arXiv preprint arXiv:2106.10404*, 2021.

[28] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[29] P. Micikevicius *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[30] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[31] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[32] T. Wang, R. M. Anwer, H. Cholakkal, F. S. Khan, Y. Pang, and L. Shao, "Learning rich features at high-speed for single-shot object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1971–1980.

[33] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 558–567.

[34] L. Liu *et al.*, "On the variance of the adaptive learning rate and beyond," 2020.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[36] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[37] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[38] S. Jastrzębski *et al.*, "Three factors influencing minima in sgd," *arXiv preprint arXiv:1711.04623*, 2017.

[39] M. C. Arya and A. Rawat, "A review on yolo (you look only one)-an algorithm for real time object detection," *Journal of Engineering Science*, 2020.

[40] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic model pruning with feedback," in *International Conference on Learning Representations*, 2020.

[41] L. Cai, Z. An, C. Yang, and Y. Xu, "Softer pruning, incremental regularization," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 224–230.